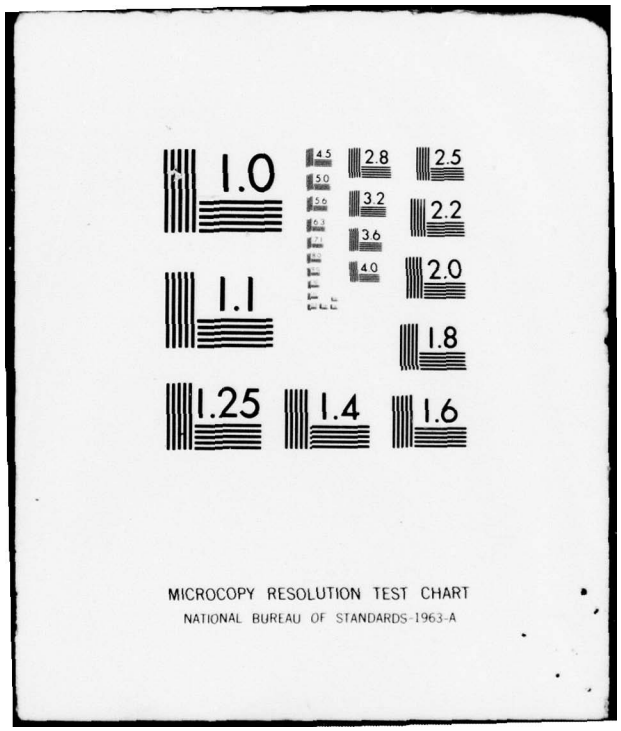
AD-A055 744 RENSSELAER POLYTECHNIC INST TROY N Y COMPUTER RESEAR--ETC F/G 9/2
DOCUMENTATION MANUAL FOR CHAP.(U)
MAY 78 K P LOEPERE
AFOSR-76-2937

UNCLASSIFIED CRL-57 AFOSR-TR-78-1039 NL

END
DATE FILMED
8 -78
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Rensselaer Polytechnic Institute

Troy, New York 12181

FOR FURTHER TRAN

(14) CRL-57

Technical Report CRL-57

(6) DOCUMENTATION MANUAL FOR CHAP .

By

(10) Keith P./Loepere

(11) May 1978 (12) 54p.

(9) Interim rept.,

(16) 2304

(17) A2

Prepared for

Directorate of Mathematical and Information Sciences
Air Force Office of Scientific Research
Air Force Systems Command, USAF
Grant (15) AFOSR-76-2937

Rensselaer Polytechnic Institute

TROY, NEW YORK 12181

409 952

ABSTRACT

This is a documentation manual for the CHAP chain processing language. CHAP is a collection of routines developed for analyzing, synthesizing, and manipulating chain-encoded line drawings. This report describes the internal operation of the CHAP routines. It is a companion volume to the CHAP User's Manual.

ACKNOWLEDGMENT

The work described here was supported in part by the Directorate of Mathematical and Information Sciences, Air Force Office of Scientific Research, under Grant AFOSR 76-2937, Professor Herbert Freeman Principal Investigator.

CONTENTS

PART 1

INTRODUCTION

CHAP is a set of routines designed to facilitate the manipulation of chain-encoded line drawings. This material highlights the operation of CHAP. To comprehend it fully, it is necessary to be familiar with the references listed below. Also, the discussions in this manual relate to the program listing of CHAP which is available upon request from Dr. H. Freeman, Rensselear Polytechnic Institute, Troy, N. Y. 12181.

Sections included herein describe the operation of the CHAP routines. The primary purpose of this manual, though, is to provide information to be used to create new implementations of CHAP and to extend CHAP in the future.

Primary References

1. K. Loepere, "CHAP User's Manual", Tech. Rept. CRL-56 ESE Dept., Rensselaer Polytechnic Institute, Troy, New York 12181.

2. H. Freeman, "Computer Processing of Line-Drawing Images", Computing Surveys, 6, (1), March 1974, 57-97.

PART 2

DATA STRUCTURES

A. Chains

A chain is stored in an integer array. There is a four-word header followed by the actual chain data. The header contains the following information:

word 1: maximum number of chain links that this array can hold

word 2: present number of links stored in this chain

word 3: 1 means that the last link in the chain is a part of a signal code; 0 means that the last link is a directional link

word 4: the index of the buffer in STATUS associated with this chain

The chain links start in the fifth word. The end-of-chain code is not stored in the chain and the link count does not count anything other than the links actually in the chain. Word 3 of the header is used when adding links to the end of the chain. This word must always be set when adding links. When this value is zero and a 4 link is added, an extra 04 must be added to change the old 0 and the new 4 to a 0404 code. This is done only when the last chain link is 0 and when this is flagged by the word 3 zero code indicating a directional link.

Links are stored as series of three binary bits. A multiple of three number of bits (30 for IBM, 36 for UNIVAC) at the end (low order bits) of each word are used to store links. The link in the highest bit positions used to store links in the

first word that is used to store chain links is numbered 1. The next three bits down correspond to link 2 and so on.

B. PLANES

PLANES is the common block that contains the binary plane. Its definition is /PLANES/ NX, NY, PLANE which appears in the CHAP block data subroutine. NX and NY are the dimensions of the binary plane; the plane is dimensioned (0: NX-1, 0: NY-1). NX must be an integral multiple of the number of bits used per word as defined in CHAPMC. PLANE is an array of bits. The binary plane is stored by columns in PLANE. Since NX is an integral multiple of the number of bits used per word, a column fits in an integral number of (consecutive) words.

C. STATUS

STATUS is the open chain status array. It is defined as /STATUS/ NUM, LENG, STATES in the block data routine were LENG is 6 and STATES is an integer array dimensioned by NUM * LENG. NUM is the maximum number of chains allowed to be open at a given time. STATES is considered to be an array which contains NUM (dimensioned 0 to NUM-1) rows each of length LENG all stored consecutively. Each row corresponds to an open chain. The contents of the words in a given row are:

word 1: the machine address of the chain corresponding to this entry

word 2: 0 if this entry does not correspond to a chain;
1 if there is a chain open for which this row is being used

word 3: the last link in the chain processed - this will either be the number of the last directional link returned by a call to GET, the number of the last link in the signal code that the last returned value consisted of or from which the last value originated (in the case of multiple link codes), or, for code 0421, the number of the link just before the start of the group of links specified if this code has not been completely returned or the last link in the 0421 code if this repeat code has just been completely processed

word 4: 0 means that either the last value returned was a directional link or that a signal code was just completed; 1 means that the last value returned was the 0 in a 0404 signal code; 2 means that a link repeat code has not been completed by the last value returned; 3 means that a group repeat code has not been finished

word 5: for case 2 in word 4 this is the link being repeated; for case 3 in word 4 this is the length of the group to be repeated

word 6: for cases 2 and 3 in word 4, this is the number of links (total) that are remaining to be returned for this signal code (codes 17, 20, and 21)

Presently, NUM is set to 4 and STATES has 24 elements.

D. WORK

/WORK/ SIZE, TEMP is the temporary workspace defined in the users program. TEMP is an integer array of dimension SIZE.

E. Constants

There are two common blocks defined in the CHAP block data routine that provide constants.

/XYCOMP/ AX, AY are the x and y (respectively) components of the link types. AX and AY are eight-element integer arrays. They contain the increments for the link types in order 0 to 7.

/CHAPMC/ NBW, NLW, EX, BLANK, DIGITS, MAXINT are all integer constants.

NBW - number of bits stored per word - This number must be a multiple of three. This is the number of bits used per word for storing both chain links and binary plane bits. The defined number of bits are stored in the low order positions of the words.

NLW - number of links stored per word - This number is equal to NBW / 3.

EX - the internal character code for an 'X' suitable for printing under an A1 format

BLANK - as above, but the character code for a blank

DIGITS - an eight element array giving the character codes for the eight octal digits in order 0 to 7

MAXINT - the maximum positive machine representable integer

PART 3

PRIMITIVE ACCESS ROUTINES

The routines in this section provide bit level access in some way. They are all machine dependent.

ADDR (ARRAY) is an integer function which returns the machine address of its (integer array) argument. For the IBM implementation, this is an assembly language routine. The UNIVAC version uses the routine LOC to provide this function.

GTLINK (BUFFER, I) is an integer function that returns the I^{th} link in BUFFER. BUFFER is the address of the first word in the chain (first word of the header). The routine computes the word index in which the I^{th} link is found. For the IBM implementation, assemble language routines are used to extract the link. UNIVAC uses the FLD function.

STLINK (LINK, BUFFER, I) is the reverse of GTLINK. The word in BUFFER where the I^{th} link is found is determined. The I^{th} link is set to the value of LINK. All other links in the chain are unaffected.

BPRINT (IW) is the binary plane print routine. This routine contains a nested set of do loops that extract the appropriate bits one at a time and put out an 'X' in the print line for each corresponding 1 bit and a blank for each 0. The bits are extracted by FLD in the UNIVAC version and by assembly language routines for IBM.

APLANE (CHAIN) adds a chain to the binary plane. It uses GET to extract the links one at a time and adds the vertices to the plane if they are within the limits of the plane. Visible

and invisible signal codes are detected and used to set a flag indicating whether or not a given in range vertex should be printed. This routine also detects the x and y coordinate specifiers and alters its running x and y location pointers.

BPLANE (CHAIN) clears the binary plane (word by word) and then uses APLANE to add the chain to it.

PRINT (CHAIN, IW) uses BPLANE to add the chain to the plane and then uses BPRINT to print the plane.

The above routines in the UNIVAC version use FLD to perform bit manipulation. The IBM version uses the three routines IAND, IOR, and SHIFT. These are all integer functions. IAND and IOR compute the bitwise and and or (respectively)of their two arguments. SHIFT (WORD, N) shifts WORD N places left circular. The shift is done on a basis of 30 bits. Bits shifted out of bit 2 are shifted back onto the right end of the word.

PART 4

DIGIT ACCESS

These routines are the standard input and output routines used by the remainder of the routines. They have a standard calling sequence so that they may be equivalenced.

A. Input

The standard input digit routine calling sequence is NAME (UNIT, INIT, CHAIN, OVER) where NAME is an integer function. This routine fetches the next digit from CHAIN or from logical unit UNIT, which ever is appropriate. If INIT is true, this call will not fetch the next digit but will reset the input. For a chain, this corresponds to the beginning. For a input unit, this corresponds to starting a new line. The next call with INIT false will get the first digit. OVER is set to true if there is no next digit; in this case, the functional result is zero.

GETDIG gets a digit from a logical unit.

CHDIG gets a digit from a chain. This is done by calling DIGCH (UNIT, INIT, CHAIN, OVER, NUM) with a NUM value of 1 (normal mode). The routine ICHDIG (VAL) is associated with this routine. It calls DIGCH with a NUM value of two. This causes DIGCH to set its chain pointer to point to the VALth link. The next call to CHDIG will then return the VAL + 1st link. The value returned by ICHDIG (and by DIGCH with mode 2) is the old value of the chain pointer (last digit number returned).

Notice that the chain or input pointers are kept in the routines. Only one chain or unit can be operated on at a time by these routines.

B. Output

The standard output digit routine calling sequence is NAME (UNIT, DIGIT, CLEAR, CHAIN). This routine adds DIGIT to the end of the chain CHAIN or the logical unit UNIT, which ever is appropriate. An error termination occurs if there is no room to hold the digit. If CLEAR is set to true, the DIGIT is ignored but the remaining digits stored within the routine will be cleared. For line routines, this means that the next digit that is sent for output will begin a new line.

PUTCH is the standard output routine. It increments the link count in the chain and adds the digit. CLEAR is not meaningful to this routine. This routine can operate on several different chains in succession without using clear since nothing is stored within the routine.

PUTDIG is the punch card output routine. It is automatically initialized to start on a new card with a sequence number of one. The CLEAR operation will reinitialize it.

NDUMB is a dummy output routine.

LISDIG is the printing routine. It calls routine DIGLIS with mode 1. Associated with this routine are three support routines that each call DIGLIS. They are: SIGLIS (UNIT) (mode 2), SPACET (UNIT) (mode 3), and SIGEND (mode 4).

DIGLIS (UNIT, DIGIT, CLEAR, CHAIN, MODE) is the actual printing routine. The normal operation of this routine is to accumulate groups of five digits and then to add them to the print line with a blank between the groups. A check is made to be sure that the group will completely fit on a line. Whenever a group

will not fit on a line, the present contents of the line are printed and a new line started for the group. CLEAR works in a similar fashion except that the last group need not contain five digits.

Whenever the MODE is not 1 for DIGLIS, the call specifies an output formatting. Mode equal to 2 is the start of a signal code specification. Any digits saved inside the routine are added to the print line (and the print line printed if necessary to provide room for the digits). Assuming that there is room on the line, a double blank followed by 04 is then added. If there is not room on the line, a new line is started with 04. The double blank is not added if the 04 will start a line, if the last entity added to the end of the line was a signal code (which ends in a double blank). Mode 3 adds a single space. Mode 4 (end of signal code) adds a double blank. These modes clear out any digits (going to a new line if necessary) and add the appropriate number of blanks unless this is the start of a new line. Mode 4 also sets a flag indicating the presence of the double blank for use if the next function is mode 2.

DUMB (UNIT) is a dummy version of SPACET. Signal code routines call a spacing function to identify signal code fields. SPACET is used for printing. DUMB would be used otherwise unless it is necessary to take certain special signal code fields into account.

OCTAL is a special output routine. It calls LOCTA (UNIT, DIGIT, CLEAR, CHAIN, MODE) with MODE equal to 1. This call causes a running total (initial value of zero) to accumulate

the digits sent to it. The digits form an octal number with the first digit sent becoming the high order digit, etc. Calling OCTIN (OUT) causes a call to LOCTA with a mode of 2 to zero the running total and to assign the old value of the total to OUT. This routine set is used to form octal numbers out of signal code fields.

PART 5

ACCESS TO GROUPS OF DIGITS

These routines access several digits at a time.

QCOMM (IN, OUT, UNIT, CHAIND, CHAINS, SPACE) is the variable length comment move routine (code 0411 . . . 04127777). UNIT, CHAIND, and CHAINS are, respectively, the logical unit, destination chain, and source chain to be used whichever are appropriate. No provision is made for moving from one logical unit to another. Movement takes place from the input using function IN to the output using subroutine OUT. Prior to calling this routine, the 0411 should have been moved. Also, a space should have been added. This routine then moves the input to the output until it finds a possible 04127777 group. If the group is found, a space is added by SPACE (unless there were no digits moved between the original 0411 and this group), 0412 is added, another blank, and 1277 then added. If the group is not the full 04127777, the digits are output and processing continues. An error occurs if the input runs out before 04127777 is encountered. No signal code end is flagged after the 04127777 is moved.

QMOVE (COUNT, IN, OUT, UNIT, CHAIND, CHAINS) also moves from the input (logical unit UNIT or chain CHAINS) using function IN to the output (logical unit UNIT or chain CHAIND) using routine OUT. The number of digits to be moved is given by COUNT. The call is ignored if count is zero.

SIGNAL (IN, OUT, MOVE, SPACE, UNIT, CHAIND, CHAINS, CODE) is the general purpose signal code move routine. Prior to the call, 04 and the two digits defining the code (given by CODE)

should have been moved. Movement is performed as for QMOVE. MOVE is the routine to perform movement of a given number of digits. SPACE is the routine to be used for spacing (or flagging) the output. This routine performs the proper grouping and spacing of all recognized signal codes and flags those it does not recognize. The end of signal code is not flagged by this routine.

PUTCHF (CHAIN, MAXNUM, NODIG, NUMB) is an output to chain routine. NUMB is checked to make sure it is not negative and is less than MAXNUM. If so, a NODIG number of digits are extracted one at a time from NUMB. The digits are extracted with the low order octal digit last. The digits extracted are added to the end of the specified chain.

MSTORE (NUMB, CHAIN, DIGITS) extracts a DIGITS number of digits (low digit last) from NUMB and adds them in order to the end of CHAIN.

LINIT (CHAIN, LINK, YES) is the first routine to look at word 3 of the header of a chain. This routine checks to see if, when directional link LINK is added to the end of the chain CHAIN, a 0404 code needs to be generated. If the last entity (if there is one) in the chain is a directional link and if it is zero and if the link to be added is a four, another 40 is added to the chain so that when the 4 is added (not by this routine), a 0404 code will result. If this addition is performed, YES is set to true.

PART 6

ADDING LINKS

CLEAR (CHAIN) sets the link count to zero and the last link type to directional in the specified chain. If the chain is found in any used entry in STATUS (ie., is open), a termination occurs.

INVIS (CHAIN) adds a 0401 code and sets the signal code end flag.

VISIBL (CHAIN) adds a 0402 code and a signal code end flag.

COLOR, ELEVAT, GREY, XCOORD, and YCOORD all of (CHAIN, VALUE) use PUTCHF to add the desired VALUE after the appropriate signal code determiner (04xy). For XCOORD and YCOORD, the value is biased by 16384 to allow positive and negative values for VALUE. In all cases, the signal code end flag is set.

POINT (CHAIN, VALUE) operates line GREY..

NODE (CHAIN, NODES, INTERS) is similar to the above except for having two fields to add by PUTCHF.

SCLIND (CHAIN, MODE, SCALE, POS) has three fields to add by PUTCHF.

ROTIND (CHAIN, ANGLE) adds a 0414 code. ANGLE is converted to radians in the range 0 to 2 * PI. The whole part of this angle is added and then five digits are extracted from the fractional part, high digit first. The digits are added in order to the chain. The signal code end flag is set.

LINK (CHAIN, LINKS, NTIM) optimizes its addition of the links. If NTIM is less than eight, the link LINKS is added the

appropriate number of times. LINIT is used to see if a 0404 code must be generated. The signal code end flag is not set unless only one link was added and that link formed a 0404 code. For NTIM values greater than seven, signal codes are generated and the signal code end flag set. A 0417 code is generated if NTIM is less than 512. For greater values of NTIM, a 0420 code is generated. The number of digits needed to represent NTIM is computed. The 0420 code is generated with a count field of this size. An error occurs if it is not possible to represent NTIM in fewer than 12 digits.

LINKSQ (CHAIN, LIST, DIM, NTIM) generates a 0421 code. The appropriate fields are filled in with DIM and NTIM (by PUTCHF) and then the links are added. The signal code end flag is set.

CHLINE (CHAIN, DELTX, DELTY) uses the Bresenham algorithm to generate a straight line approximation. A pseudo quadrant number is determined for the endpoint. (1 means octant 1, 2 is octant 2, 3 is 4, 4 is 3, 5 is 8, 6 is 7, 7 is 5, and 8 is 6) The octant number establishes the link types in the approximation. The absolute values of DELTX and DELTY are determined and ordered to find the limits of the approximation loop. The standard Bresenham algorithm is then used to determine the link sequence. Unless only one link is added and LINIT produces a 0404 code, the signal code end flag is not set. (Reference: Bresenham, J. E., "Algorithm for computer control of a digital plotter", IBM Systems J., (4), 1965, pp. 25-30)

PUT (CHAIN, LINK, FLAG) contains a mixture of functions. Arguments are checked in all cases. If link is in the range 0 to 7 (directional link), it is added, possibly with a 0404 signal code generated by LINIT. For signal code values of LINK, the appropriate case is selected and fields, if necessary, are added by PUTCHF. An unrecognizable value for link results in an error.

Two routines associated with PUT are PPUT (CHAIN, LINK, FLAG) and PCLOSE (CHAIN). These routines pack link repeat signal codes. PPUT keeps a count (initially zero) of the number of consecutive occurrences of a link. When a different link or a signal code occurs, LINK is called to pack the number of occurrences found. The new signal code is output or the new link is recorded with a count of one. PCLOSE causes the remaining link and count in the routine to be cleared out by LINK. PUT is used to add signal codes. PPUT is equivalent to PUT except that it maintains this internal counter which prevents this routine from being usable for more than one chain at a time. PCLOSE is necessary to allow the routine to be used for a different chain. PCLOSE operates by calling PPUT with a link value of -1; PUT would flag an error for this argument.

PART 7

INPUT AND OUTPUT

INPUT (UNIT, CHAIN) is the actual chain input routine. A check is first made to see if the chain is open (present in a used entry in STATUS). Given that it is not open, digits are read in from the desired unit one at a time and added to the end of the chain. When a 0 is encountered, the next digit is checked for the possibilty of a signal code. If the next digit is not a 4, the 0 is output and processing resumes. When a code is found, SIGNAL is used to add it to the chain. A signal code of 0400 is not added to the chain and encountering such a code ends the input process. The signal code end flag is set appropriately.

OUTPUT (UNIT, CHAIN), the punch routine, is nearly identical to INPUT. In this routine, however, all signal codes including 0400 are punched. Encountering a 0 punches it but causes a check for a possible signal code. When a signal code is found, either SIGNAL is used to output it or (in the case of 0400), the code is punched directly. Encountering the end of the chain is recognized as end of chain and causes a 0400 code to be punched.

LIST (UNIT, CHAIN), the listing routine, differs from OUTPUT only in that calls are made to SIGLIS and SIGEND, when appropriate, to format the output.

Notice that output routines reconstruct the end of chain signal code that the input routine threw away. The input routine does not consider the 0400 code when deciding to set the end signal code flag since this code is not at the end of the chain.

PART 8

SEGMENTING

NMOVE (CHAIND, CHAINS, SIG1, SIG2, OUT, END, SIG, LAST) moves a chain segment. Once started, this routine will move from CHAINS to CHAIND using OUT as its output routine, that part of CHAINS up to, but not including, a signal code of type SIG1 or SIG2. Links are extracted one at a time from CHAINS. If the end of chain is encountered in doing this, END is set and the routine returns. If the digit is not 0, it is output. For a digit of 0, a check is made for a signal code. The 0 is output if the next digit is not 4. A 04 combination, flagging a signal code, causes the next two digits to be fetched and checked. If the code is not SIG1 or SIG2, SIGNAL is used to move the signal code fields once the 04xy digits are moved. When SIG1 or SIG2 is found, the digits 04xy are not added to the chain and processing stops. SIG is set to the signal code determiner that was detected. Also, LAST is set to a 1 (0 otherwise) if the last entity moved to CHAIND (not counting the terminating signal code) was a signal code.

VSEG (CHAIND, CHAINS, LIMIT1, LIMIT2, SIGY, TEMP, N) moves all segments of a given value. Two ranges of values are established to cover the two possible value divisions. If the value 0 (default attribute value) is in the set to be moved, a signal code of type SIGY and with a value field of zero (consisting of N digits) is generated. The routine switches its state back and forth between actual transfer (using PUTCH) and just passing the input (using NDUMB). The initial state depends

on whether or not Q is in the set of values to be moved. NMOVE is used to move all portions up to the occurrence of the signal code SIGY. When SIGY is found, the temporary array TEMP (N) is used to accumulate the digits in the signal code field. The value is checked to see if it is in range. If so, a signal code of type SIGY with the value accumulated is produced. Otherwise, the state is changed to pass mode and the signal code is not produced in CHAIND.

NTHSEG (CHAIND, CHAINS, LIMIT1, LIMIT2, SIG1, SIG2, NSIG, INIT) moves segments by number. SIG1 and SIG2 are the signal codes of opposite attributes (such as visible and invisible). Movement takes place only for segments of type SIG1 with numbers in range. Movement occurs possibly in two phases; if the limits specify a double range, the move portion of this routine is used twice with different limits. The state of this routine switches back and forth as the chain segments either turn into segments with attributes SIG1 or SIG2. If INIT is 1 (specifying that the default attribute is type SIG1) and segment 1 is in range, NSIG (CHAIND, CHAINS) is called to generate the initial signal code corresponding to the default attribute. INIT is zero if this is not to be done. From then on, starting with the appropriate initial state, segments are either moved or passed (PUTCH or NDUMB) until SIG1 or SIG2 is encountered by NMOVE. If the chain becomes of type SIG1, the segment count is incremented. The limits are checked to see if this next range should be moved. If code SIG2 is encountered, the segment counter is not incremented and the state changes to pass. For segments with a move state,

the signal code that is appropriate is generated and SIGNAL is used to move the as yet unpassed fields to the output chain. Movement will then proceed until the SIG2 code is encountered or a SIG1 code causes the segment counter to be incremented out of range.

Notice that the routine NMOVE produces a flag indicating the last entity moved (signal code or directional link). VSEG and NTHSEG put this value into the chain field for this whenever the move was an actual transfer. Also, generation of signal codes, either initial or as a result of transfer, are also so flagged in the signal code end flag in the chain.

MARKER (CHAIND, CHAINS, LIMIT1, LIMIT2) is a separate routine. If LIMIT1 is out of range for markers (ie., movement starts at beginning), a check is made for the possible need of a 0404 code when the two chains are joined and the routine starts movement from the beginning. Otherwise, CHAINS is passed until NMOVE finds a marker which this routine recognizes as LIMIT1. When this is found, a signal code with this value is added to CHAIND. Movement then occurs, using NMOVE, until a marker is found. The marker code is always added to the end of CHAIND. However, if the marker is equal to LIMIT2, the operation of this routine ends. The signal code end flag is set whenever a signal code (including the marker codes) is added to CHAIND.

PART 9

SEGMENT ROUTINES

COLORD, ELATED, and GREYD all of (CHAIND, CHAINS, LIMIT1, LIMIT2) use VSEG to move the desired segments. They also check the limits to be sure they are valid for the type of code being found.

INVSEG, VISSEG, NCOLOR, NELEV, and NGREY all of (CHAIND, CHAINS, LIMIT1, LIMIT2) and their corresponding initial signal code routines NINV (a dummy routine since it is never called), NVIS, NCOLS, NELVS, and NGRYS use NTHSEG to move the desired segments.

PART 10

SEQUENTIAL ACCESS

INITXY (CHAIN, X, Y) scans for the last x and y coordinate specifiers before the first link. X and Y are given initial values of 0. When an actual link is found, the routine returns. Otherwise (signal codes), the signal code is checked to see if it can generate a link. Those that definitely can not are passed. Those that definitely do cause the routine to return. For link repeat codes, if the count is zero, the code is passed. For group repeat codes, if the count is zero, the input pointer is repositioned past the link group; if the group length is zero, the code is passed. If these codes do represent links, they cause the routine to return. Whenever an x or y coordinate specifier is encountered, the value is computed and the appropriate value is assigned. The last values so assigned will be returned.

OPEN (CHAIN) scans STATUS. If this chain is found in an active entry, an already open error occurs. Assuming that the chain is not open and that there is room for it, the STATUS entry will be set as active and to be used for this chain. The chain field is set to the entry number in STATUS. The STATUS entry is also set so that GET will start from the beginning.

CLOSE (CHAIN) looks for the (assuming there is one) used entry in STATUS associated with this chain and sets it to be not used.

GET (CHAIN, FIRST, SECOND) first checks to see if the chain is open by matching the field in the chain with the STATUS entry. If the chain is not flagged as being open by the

corresponding entry, 10, 0 are returned for FIRST and SECOND. For an open chain, the action depends on the value of the fourth entry in the STATUS entry. A value of one will return the extra 4 in the 0404 code and set the entry back to 0 for normal processing on the next call. For mode 2, the link is returned, the count decremented, and if 0, the STATUS entry set to zero for the next time. A mode of 3 will cause the appropriate link in the group to be returned, the total count to be decremented, and, if 0, the mode to be reset and the last link encountered pointer to be updated to the end of the code group. If the mode is zero, this implies normal processing. A normal link is returned and the link counter updated if the next entry is a link. For field value signal codes, OCTAL is used to produce the octal number and the link counter is appropriately updated. Signal codes that produce no output are passed. Link or group repeat codes cause the mode to be set accordingly and the STATUS fields set. Operation then proceeds as if the routine had been entered in that mode. Link or group repeat codes that do not specify links are skipped. Notice that the last link encountered indicator is set by the result of ICHDIG and that ICHDIG is used to position the input to the desired point.

PART 11

ARRAYS OF LINKS OR VERTICES

ARRAY (CHAIN, LIST, N, L, OVER) is a simple loop that uses GET to extract links. Signal codes are ignored. Each link is added to LIST with an appropriate check for array overflow made.

INVERT (CHAIN, LIST, N, L, OVER) first calls ARRAY with the same arguments. If this call was successful, elements from opposite ends of the filled in portion of LIST are interchanged forming their inverse form as they are moved. If there is an odd number of elements (one element with no element to interchange with) this element is inverted separately.

CHPAX (CHAIN, LIST, N, L) operates just like ARRAY except that overflow is an error and that the links are converted to PAX form before inserting into the LIST. The element beyond the last link filled in is set to zero.

VERTEX (CHAIN, XCOORD, YCOORD, N, L, OVER) sets the first values in XCOORD and YCOORD by using INITXY. From then on, each link produces a new pair of x and y values appropriately incremented beyond the last pair of values. All signal codes except x and y coordinate specifiers are ignored. These last two codes update the corresponding element of the pair of the last vertex computed. A check is made for array overflow before entering a new pair of values.

PART 12

ANGLES, DISTANCES, AND EXTENTS

ANGLE (CHAIN, NP1, NP2, ANGLE)first orders NP1 and NP2 remembering what node was specified first. Links are extracted one at a time with the x and y increments of these links being accumulated. When NP1 is encountered, the x and y coordinates are saved. The scan then continues for NP2. Once found, the angle is computed. If the nodes had to be reversed, the angle is reversed also. X and y coordinate specifiers are the only signal codes that are used. A special check is made if the node desired is the last node of the chain.

MAXMIN (CHAIN, XY) takes each link, one at a time, and uses it to update its running x and y pointers. X and y coordinate specifiers are also used. After each link has been seen, the new x and y values are compared against the old maximum and minimum values to decide the new running maximum and minimum values.

PDIST (CHAIN, NODE1, NODE2, DIST) orders NODE1 and NODE2. The chain is scanned for the low node number. The coordinates of this point are remembered. In determining these coordinates, links update running x and y counters and x and y coordinate specifiers set these counters. Once the high numbered node is found, the distance can be determined from the x and y change between this point and the point remembered.

WHEX (CHAIN, ITYPE, W) has an array that gives, for each link and each extent type, the increment that this link gives in the specified direction. Each link is taken one at a

time and used to update the present extent in the appropriate direction. A running maximum and minimum extent are maintained. The final value is the difference of the maximum and minimum encountered extent.

LENGTH (CHAIN, CHL, LCF) has two running counters to remember the number of even and odd links encountered. For each link, it is decided whether or not it should be included in the count (depending on the value of LCF and on the presence of visible or invisible signal codes encountered). The calculated length is derived from the number of odd and even lengths included.

RESID (CHAIN, LRES1, NLRES1, LRES2, NLRES2) first determines the end point of the chain by keeping a running x and y counter which are incremented by each link and which are set by x and y coordinate specifiers. The initial and final coordinates are used to determine the displacement of the chain and the pseudo quadrant in which this displacement occurred (1 means octant 1, 2 is 2, 3 is 8, 4 is 7, 5 is 4, 6 is 3, 7 is 5, and 8 is 6). The link types for each of these octants is looked up and the number of each link computed.

PNTCND (CHAIN, XP, YP, DMAX, DMIN, LMAX, LMIN) also keeps running x and y coordinate pointers. X and y coordinate specifiers update these pointers as well as links. After each link, the (square of) the distance between the endpoint of the link and (XP, YP) is computed and compared against the running maximum and minimum distances (initial value is the distance to the origin of the chain). If this new distance becomes either

the new maximum or the new minimum, the vertex number is also recorded in the appropriate variable. The correct distances are computed at the end.

LNCHD (CHAIN, X1, Y1, X2, Y2, CLDMAX, CLDMIN, JMAX, JMIN) records distances relative to the line specified. The points on the line are used to compute, for each link, the corresponding increment in perpendicular distance to the line that this link produces. Starting with an initial distance computed from the origin, each link is used to update the distance and possibly update the maximum and minimum recoreded distances. When a new minimum or maximum is encountered, the vertex number is recorded. X and y coordinate specifiers also update the distance but are not used to determine maximum and minimum.

PART 13

AREAS AND MOMENTS

ECAREA (CHAIN, S) uses the standard area formula. A running y counter which is incremented by links and set by y coordinate specifiers is kept.

CENTRD (CHAIN, X, Y) uses ECAREA to find the area and MOM1 to find the x and (negative) y axis moments. The centroid coordinates are the appropriate ratios of these values.

MOM1 and MOM2 (CHAIN, DEGREE, MOMENT) use MOM1A and MOM2A, respectively, to find the given moments. Values for the translated origin are taken from INITXY.

MOM1A and MOM2A (CHAIN, DEGREE, INITX, INITY, MOMENT) are identical except for the actual formula used to determine the moment. They both keep a running y counter which is incremented by each link. The y value is the distance to the axis. The initial value of y is appropriatelt computed for each axis type. For each link, the x and y increments relative to the axis are computed. Given the x and y increments and the y value, the increments to the moments are computed and summed.

LMOM1 (CHAIN, X1, X2, Y1, Y2, FMNT) computes the first moment as the area times the perpendicular distance from the line to the centroid of the chain.

LMOM2 (CHAIN, X1, X2, Y1, Y2, SMNT) uses MOM2A to compute the moment if the line axis is vertical. Otherwise, the increments in x and y relative to the line are computed for each link type. Each link increments the present distance to the line counter (which is initialized by the origin of the chain). The x and y increments and distance are used to increment the moment.

PART 14

SPECIAL FUNCTIONS

AUTO (CHAIN, NA, ACORR, M) fills the temporary workspace with the links of the chain. For each shift value from 1 to the number of links / 2 (a shift of 0 is automatically a correlation value of 1), a do loop computes the standard correlation function.

CROSS (CHAIN, CHAIN2, CORR, J) first uses ARRAY to put the links of chain 2 into the temporary workspace. These links are then moved to the end to make room for ARRAY to place the links of the first chain. The correlation value is computed in the standard way.

BAYPEN (CHAIN, MAXBAY, MAXPEN, BAYAR, PENAR, H) is a mixture of area and distance functions. RESID is used to determine the endpoints of the segment. INITXY provides the starting point. The distance between them is then computed. The operation then proceeds in a similar fashion to LNCHD and LMOM2. The increments in x and y relative to the line for each link is computed. A running distance is kept. For each link, a new distance is computed and considered in the maximum bay or peninsula depending on whether or not the distance is positive or negative. A flag is kept to remember on which side of the chord the present point is on. If the side does not change, the area for that side is incremented by the link. When the side does change, beside changing the flag, this link (which crosses the chord) adds an increment of area to both sides of the chord. The area of the two triangles formed by the link, the chord, and the new and old distance vectors are added to the appropriate side areas. Notice

that the distance is incremented before the area is computed so the area formula is modified slightly.

CENPRO (CHAIN, PROFL, DIM, N) first finds the centroid which is rounded to the nearest integers. For each link, the residue distance to the centroid is computed and recorded in PROFL with an appropriate check for overflow. The index of the maximum distance is recorded by keeping a running maximum (absolute) distance and a running index number of the maximum distance. Also, as the residue calculations proceed, the displacements from the centroid and the x and y increments of the link are used to determine the direction of rotation and the sign of the profile distance. Once all vertices are computed, the profile array is shifted to bring the maximum distance to the first position. Also, normalization occurs during this shift. The shift consists of taking the maximum element, moving it to position one, moving the old element that was in position one to where it belongs, etc. If this does not move all elements (shift was not relatively prime to the number of elements), a new subset of elements one greater in index is then moved until all have been shifted.

XPROFL (CHAIN, FLAG, PROFL, N, INDEX) fills the temporary array space with the links of the chain. If FLAG is false, the chain links are rotated to bring the left edge to the right. PROFIL is then used to compute the profile.

YPROFL (CHAIN, FLAG, PROFL, N, INDEX) also fills the temporary array with the links and rotates them so as to allign the desired edge with the right edge of an encasing rectangle.

PROFIL computes the profile.

PROFIL (PROFL, DIM, INDEX, LL) computes profiles. LL is the number of links in the temporary workspace to be used. PROFL, DIM, and INDEX correspond to PROFL, N, and INDEX in the XPROFL and YPROFL headers, all respectively. The extents of the encasing rectangle are determined from the minimum and maximum extents along with determining twice the area. The area is completed by a straight line from (0, 0) to (x, y) (the end of the chain). The greatest x (and the number of the link arriving there) on the lower border is found. If the chain encircles in the wrong direction, it is inverted and the maximum lower edge x link number is also changed. For each link from the maximum lower edge x and up along the chain (possibly wrapping around for a closed chain), the x and y increments are examined. Upward directed links establish new profile values. Sideward directed links update only the second value of the old pair. If a downward directed link is also directed outward, it updates the second value of the pair down a value (a later upward directed link will set these values again). Downward and inward directed links force the routine into a different mode where it will examine links until they move up again to the value of y at which the curve started moving down. No action is taken until the curve is back out of the peninsula or until the curve breaks out and is moving outward. The routine stops when the curve reaches the top of the encasing rectangle. The end profile pair values are given the lengths of the encasing rectangle where appropriate.

MATCH (XX, YY, PROF1, N1, PROF2, N2) matches the output of PROFIL. It maximizes the area of overlap of encasing rectangles. The operation takes place in two parts. In the second, one chain is shifted sufficiently to overlap another completely. In this case, the area of overlap is given by the length of the regions in which the profile lengths overlap times the distance one profile can be shifted into another. The minimum distance between profile values (starting from an absolute maximum of the sum of the encasing rectangle sides) is found. A different loop finds this distance depending on which chain overlaps which in length. The first phase of the routine is for the case where the lateral shift values being considered are insufficient to shift one chain completely overlapping the other. This portion checks for both the case were chain 2 is shifted in from above and from below. The maximum possible inward shift (minimum distance between profile values) is found and the area of overlap (inward shift times length of overlap) determined. As the maximum area is found, the area is saved in a running maximum area value and the inward and lateral shifts corresponding to this value are saved.

SUBCH (CHAIND, CHAINS, VERT1, VERT2) looks for VERT1 by passing everything until it is found. Everything from then on until VERT2 or the end of the chain is PUT onto CHAIND. Links cause the vertex counter to be incremented. Non-links are transfered but do not increment the vertex counter.

INTERS (CHAIN1, CHAIN2, J1, ITYPE, INLCHA, INLCHB, NUMINT) uses the method of overlapping rectangles to minimize the number of tests that must be made. The links of both chains are

found and used to establish two arrays of x and y coordinates at opposite ends of the temporary array space. Another array is created to hold the vertex numbers in both chains that are within the overlapping rectangles. The program iteratively examines all vertices in the overlapped area of the two chains and determines those vertices of the two chains still within that region. Given this new subset of vertices, the rectangular area in which the vertices of the two chains lie is determined and overlapped to limit the region in which possible intersections occur. Once this has been done to the point at which no further reduction is possible, the intersections are found. All vertices left in chain 1 are checked against all vertices in chain 2 for a match (intersection type 1). If there is no match, the vertices are checked with the next vertex in the chain to see if they form a unit rectangle (type 2 intersection). A check is made for the case where the next vertex is outside of the rectangle. (Reference: Freeman, H., "Techniques for the Digital Computer Analysis of Chain-Encoded Arbitrary Plane Curves", Proc. National Electronics Conference, Chicago, Illinois, vol. 17, pp. 421-432, October 1961.)

POLYGN (CHAIN, XCOORD, YCOORD, ICL, TOL, IC, JJ, K) uses temporary workspace as the open stack. The x and y coordinates are generated in XCOORD and YCOORD. For a closed chain, the vertices with the maximum and minimum x values are found and the minimum is added to open and closed (ICL) and the maximum is added to open. An open chain has the first vertex put in closed and the last in open. Depending on the slope of the chord that is

being considered as an approximation, either the vertical or horizontal distance to the chain is used to approximate the true perpendicular distance. The vertex with the maximum distance is recorded and the true distance evaluated. If this distance is greater than the tolerance, the new vertex is added to open and the process repeats. Otherwise, the chord is a good approximation and so the open vertex is moved to close. Processing continues until no more open vertices remain (all chords have been found satisfactory). (Reference: Ramer, U., "An Iterative Procedure for the Polygonal Approximation of Plane Curves", Computer Graphics and Image Processing, (1972), I, pp. 244-256)

ROSCAL (CHAIND, CHAINS, ANGLE, XSCALE, YSCALE) extracts links from CHAINS and approximates them with straight lines to form CHAIND. Signal codes encountered are moved to CHAIND. X and y coordinate specifiers are transformed relative to the origin and are put into CHAIND. For each link, the new x and y coordinates after scaling and rotating (in that order) about the origin are computed (relative to the start of the chain). Starting from the end of the last link generated, other links are generated to approximate a straight line to the new point. This procedure is an iterative procedure that finds the slope of the straight line to the new point and determines where a new link would fall relative to this line. Links generated will approximate the transformed chain only on a link by link basis and only as well as the underlying grid allows the links to end at a given point.

PART 15

CREATING A VERSION OF CHAP

This section summarizes the actions necessary to create a working version of CHAP for a given computer.

A. Block Data Routine

A set of values must be chosen for NX and NY, the size of the binary plane. NX must be an integral multiple of the number of bits per word to be stored. Also, PLANE must be defined to be of size NX * NY / number of bits per word to be used.

CHAPMC must be carefully checked. NBW must be set to the number of bits per word to be used. This must be equal to three times NLW, the number of links to be stored per word. The definitions of EX, BLANK, and DIGITS may have to be changed depending on the manner in which the given FORTRAN compiler expects definitions of character constants. Finally, MAXINT must be set to the highest machine representable positive integer.

B. Bit Manipulating Routines

Probably the easiest routine to write is ADDR. This routine must return the address of its argument. The argument will always be an integer array. The value returned must be an address assigned in some unique way. Two chains that occupy different areas of memory must be given different address values and a given chain must always be given the same resultant address value. If two chains are identical but are different memory arrays, they have different values returned by this function. (This is to say that returning the address relative to the start

of the routine as opposed to the start of the core load or something to that effect is not acceptable).

The best model for the other routines that need to be modified is the UNIVAC version. In this version, there are two routines that fetch a set of bits from a word (GTLINK and BPRINT) and two that sore into a word (STLINK and APLANE). The UNIVAC fersion uses FLD (i, k, e) which is both a function and a pseudo-variable. I is the first bit to be used (where 0 is the leftmost bit in a UNIVAC word), k is the number of bits to use, and e is either the word to fetch from or the word to store into. A similar function and a subroutine which operates like this pseudo variable must be written. Watch out for the bit numbering in the computer in question; the expressions for the bits to be used in the UNIVAC expressions may have to be modified if the numbering of the bits is different.

There are no other known problems. Generally, if CHAP compiles on a given computer, it will run and adjust itself to the configuration.

If the FORTRAN in question does not pass the address of, for example, A (1, 3), but instead creates a temporary, this should be noted in the user's manual. This means that arrays of chains are not permitted in this FORTRAN.

PART 16

CALLING STRUCTURE

A. External References

The following is a list of all external references made by routines in the IBM version of CHAP. The UNIVAC version differs only in the machine dependent routines.

ANGLE: CLOSE GET OPEN XYCOMP

ARRAY: CLOSE GET OPEN

AUTO: ARRAY WORK

BAYPEN: CLOSE GET INITXY OPEN RESID XYCOMP

BPLANE / APLANE: CHAPMC CLOSE GET INITXY IOR OPEN PLANES SHIFT XYCOMP

CENPRO: CENTRD CLOSE GET INITXY OPEN XYCOMP

CENTRD: ECAREA MOM1

CHDIG: DIGCH

CHLINE: LINIT PUTCH

CHPAX: CLOSE GET OPEN

CLEAR: ADDR STATUS

CLOSE: ADDR STATUS

COLOR: PUTCH PUTCHF

COLORD: VSEG

CROSS: ARRAY WORK

DIGCH: GTLINK

DIGLIS: CHAPMC

ECAREA: CLOSE GET INITXY OPEN XYCOMP

ELATED: VSEG

ELEVAT: PUTCH PUTCHF

```
GET:    ADDR   CHDIG  DUMB   GTLINK ICHDIG NDUMB  OCTAL  OCTIN
        QMOVE  SIGNAL STATUS
GETDIG: CHAPMC
GREY:   PUTCH  PUTCHF
GREYD:  VSEG
GTLINK: CHAPMC IAND   SHIFT
ICHDIG: DIGCH
INITXY: CHDIG  DUMB   ICHDIG NDUMB  QMOVE  SIGNAL
INPUT:  ADDR   DUMB   GETDIG PUTCH  QMOVE  SIGNAL STATUS
INTERS: ARRAY  INITXY WORK   XYCOMP
INVERT: ARRAY
INVIS:  PUTCH
INVSEG: NINV   NTHSEG
LENGTH: CLOSE  GET    OPEN
LINIT:  GTLINK PUTCH
LINK:   LINIT  MSTORE PUTCH
LINKSQ: MSTORE PUTCH
LISDIG: DIGLIS
LIST:   CHDIG  LISDIG QMOVE  SIGEND SIGLIS SIGNAL SPACET
LMOM1:  CENTRD ECAREA
LMOM2:  CLOSE  GET    INITXY MOM2A  OPEN   XYCOMP
LNCHD:  CLOSE  GET    INITXY OPEN   XYCOMP
MARKER: CHDIG  GTLINK NDUMB  NMOVE  PUTCH
MAXMIN: CLOSE  GET    INITXY OPEN   XYCOMP
MOM1:   INITXY MOM1A
MOM1A:  CLOSE  GET    OPEN   XYCOMP
MOM2:   INITXY MOM2A
```

```
MOM2A:  CLOSE  GET    OPEN   XYCOMP
MSTORE: PUTCH
NCOLOR: NCOLS  NTHSEG
NCOLS:  PUTCH
NELEV:  NELVS  NTHSEG
NELVS:  PUTCH
NGREY:  NGRYS  NTHSEG
NGRYS:  PUTCH
NMOVE:  CHDIG  DUMB   QMOVE  SIGNAL
NODE:   PUTCH  PUTCHF
NTHSEG: ADDR   CHAPMC CHDIG  DUMB   NDUMB  NMOVE  PUTCH  QMOVE
        SIGNAL
NVIS:   PUTCH
OCTAL:  LOCTA
OCTIN:  LOCTA
OPEN:   ADDR   STATUS
OUTPUT: CHDIG  DUMB   PUTDIG QMOVE  SIGNAL
PCLOSE: PPUT
PDIST:  CLOSE  GET    INITXY OPEN   XYCOMP
PNTCND: CLOSE  GET    INITXY OPEN   XYCOMP
POINT:  PUTCH  PUTCHF
POLYGN: CLOSE  GET    INITXY OPEN   WORK   XYCOMP
PPUT:   LINK   PUT
PRINT / BPRINT:       BPLANE CHAPMC IAND   PLANES SHIFT
PROFIL: WORK   XYCOMP
PUT:    LINIT  PUTCH  PUTCHF
PUTCH:  STLINK
```

```
PUTCHF: PUTCH
PUTDIG: CHAPMC
RESID:  CLOSE  GET    INITXY OPEN   XYCOMP
ROSCAL: ADDR   CLOSE  GET    INITXY OPEN   PCLOSE PPUT   XYCOMP
ROTIND: PUTCH
SCLIND: PUTCH  PUTCHF
SIGEND: DIGLIS
SIGLIS: DIGLIS
SIGNAL: QCOMM
SPACET: DIGLIS
STLINK: CHAPMC IAND   IOR    SHIFT
SUBCH:  ADDR   CHAPMC CLOSE  GET    OPEN   PCLOSE PPUT
VERTEX: CLOSE  GET    INITXY OPEN   XYCOMP
VISIBL: PUTCH
VISSEG: NTHSEG NVIS
VSEG:   ADDR   CHAPMC CHDIG  DUMB   NDUMB  NMOVE  PUTCH
WHEX:   CLOSE  GET    OPEN
XCOORD: PUTCH  PUTCHF
XPROFL: ARRAY  PROFIL WORK
YCOORD: PUTCH  PUTCHF
YPROFL: ARRAY  PROFIL WORK
```

B. Referencing Modules

This list, again for the IBM version, lists the modules that reference a given other module.

```
ADDR:    CLEAR  CLOSE  GET    INPUT  NTHSEG OPEN   ROSCAL SUBCH
         VSEG
ARRAY:   AUTO   CROSS  INTERS INVERT XPROFL YPROFL
BPLANE:  PRINT
CENTRD:  CENPRO LMOM1
CHAPMC:  BPLANE / APLANE      DIGLIS GETDIG GTLINK NTHSEG PRINT /
         BPRINT PUTDIG STLINK SUBCH  VSEG
CHDIG:   GET    INITXY LIST   MARKER NMOVE  NTHSEG OUTPUT VSEG
CLOSE:   ANGLE  ARRAY  BAYPEN BPLANE / APLANE      CENPRO CHPAX
         ECAREA LENGTH LMOM2  LNCHD  MAXMIN MOM1A  MOM2A  PDIST
         PNTCND POLYGN RESID  ROSCAL SUBCH  VERTEX WHEX
DIGCH:   CHDIG  ICHDIG
DIGLIS:  LISDIG SIGEND SIGLIS SPACET
DUMB:    GET    INITXY INPUT  NMOVE  NTHSEG OUTPUT VSEG
ECAREA:  CENTRD LMOM1
GET:     ANGLE  ARRAY  BAYPEN BPLANE / APLANE      CENPRO CHPAX
         ECAREA LENGTH LMOM2  LNCHD  MAXMIN MOM1A  MOM2A  PDIST
         PNTCND POLYGN RESID  ROSCAL SUBCH  VERTEX WHEX
GETDIG:  INPUT
GTLINK:  DIGCH  GET    LINIT  MARKER
IAND:    GTLINK PRINT / BPRINT       STLINK
ICHDIG:  GET    INITXY
INITXY:  BAYPEN BPLANE / APLANE      CENPRO ECAREA INTERS LMOM2
         LNCHD  MAXMIN MOM1   MOM2   PDIST  PNTCND POLYGN RESID
         ROSCAL VERTEX
IOR:     BPLANE / APLANE      STLINK
LINIT:   CHLINE LINK   PUT
```

```
LINK:    PPUT
LISDIG:  LIST
LOCTA:   OCTAL  OCTIN
MOM1:    CENTRD
MOM1A:   MOM1
MOM2A:   LMOM2  MOM2
MSTORE:  LINK   LINKSQ
NCOLS:   NCOLOR
NDUMB:   GET    INITXY MARKER NTHSEG VSEG
NELVS:   NELEV
NGRYS:   NGREY
NINV:    INVSEG
NMOVE:   MARKER NTHSEG VSEG
NTHSEG:  INVSEG NCOLOR NELEV  NGREY  VISSEG
NVIS:    VISSEG
OCTAL:   GET
OCTIN:   GET
OPEN:    ANGLE  ARRAY  BAYPEN BPLANE / APLANE      CENPRO CHPAX
         ECAREA LENGTH LMOM2  LNCHD  MAXMIN MOM1A  MOM2A  PDIST
         PNTCND POLYGN RESID  ROSCAL SUBCH  VERTEX WHEX
PCLOSE:  ROSCAL SUBCH
PLANES:  BPLANE / APLANE      PRINT / BPRINT
PPUT:    PCLOSE ROSCAL SUBCH
PROFIL:  XPROFL YPROFL
PUT:     PPUT
PUTCH:   CHLINE COLOR  ELEVAT GREY   INPUT  INVIS  LINIT  LINK
         LINKSQ MARKER MSTORE NCOLS  NELVS  NGRYS  NODE   NTHSEG
```

```
        NVIS   POINT  PUT    PUTCHF ROTIND SCLIND VISIBL VSEG
        XCOORD YCOORD
PUTCHF: COLOR  ELEVAT GREY   NODE   POINT  PUT    SCLIND XCOORD
        YCOORD
PUTDIG: OUTPUT
QCOMM:  SIGNAL
QMOVE:  GET    INITXY INPUT  LIST   NMOVE  NTHSEG OUTPUT
RESID:  BAYPEN
SHIFT:  BPLANE / APLANE      GTLINK PRINT / BPRINT       STLINK
SIGEND: LIST
SIGLIS: LIST
SIGNAL: GET    INITXY INPUT  LIST   NMOVE  NTHSEG OUTPUT
SPACET: LIST
STATUS: CLEAR  CLOSE  GET    INPUT  OPEN
STLINK: PUTCH
VSEG:   COLORD ELATED GREYD
WORK:   AUTO   CROSS  INTERS POLYGN PROFIL XPROFL YPROFL
XYCOMP: ANGLE  BAYPEN BPLANE / APLANE      CENPRO ECAREA INTERS
        LMOM2  LNCHD  MAXMIN MOM1A  MOM2A  PDIST  PNTCND POLYGN
        PROFIL RESID  ROSCAL VERTEX
```

PART 17

CHAP TAPE FORMAT

The CHAP tape contains a variety of information. Each section is headed with a card of the form: // description. The sections on the CHAP tape with line numbers are:

IBM block data routine (2-18)

IBM assembly language routines (20-110)

IBM machine dependent routines (112-237)

UNIVAC block data routine (239-255)

UNIVAC machine dependent routines (257-380)

CHAP (382-4049)

test set 1 (4051-4130)

input to test set 1 (4132-4135)

output from test set 1 (4137-4248)

test set 2 (4250-4339)

input to test set 2 (4341-4346)

output from test set 2 (4348-4460)

programming example (4462-4499)

input to programming example (4501-4506)

output from programming example (4508-4600)

The test sets included test all of the routines in their basic mode of operations. They are not an exhaustive test set. However, comparing the output when run against the desired output provided shows whether or not the source is intact and whether or not the FORTRAN in question functions in the desired manner. (Units 5 and 6 are used for input and output. These numbers may need to be changed to run at a given installation.)

PART 18

ROUTINE INDEX

WHEX: 25, 40

WORK: 5, 43

XCOORD: 14, 40

XPROFL: 30, 40

XYCOMP: 5, 43

YCOORD: 14, 40

YPROFL: 30, 40

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 78-1039	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DOCUMENTATION MANUAL FOR CHAP		5. TYPE OF REPORT & PERIOD COVERED Interim
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Keith P. Loepere		8. CONTRACT OR GRANT NUMBER(s) AFOSR 76-2937
9. PERFORMING ORGANIZATION NAME AND ADDRESS Rensselaer Polytechnic Institute Electrical and Systems Engineering Department Troy, New York 12181		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research/NM Bolling AFB, Washington, DC 20332		12. REPORT DATE May 1978
		13. NUMBER OF PAGES 52
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

line-drawing processing
image processing
pattern recognition
cartography
computer graphics
graphics languages

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This is a documentation manual for the CHAP chain processing language. CHAP is a collection of routines developed for analyzing, synthesizing, and manipulating chain-encoded line drawings. This report describes the internal operation of the CHAP routines. It is a companion volume to the CHAP User's Manual.

DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)